6502 Assembly

James Riley Dorough

Liberty University

March 5th, 2021

This lab introduces assembly languages as a base for understanding the operation of CPU functionality. Specifically, 6502 assembly is used in this document. This paper details the 6502-assembly code used to program simple addition and simple subtraction functions. To demonstrate the functionality of this simple program, this paper includes multiple images of the deconstruction of the code and the memory locations post execution to accompany the written descriptions and research.

To understand how the 6502 assembly language works, you first have to understand the idea of opcodes. This lab uses a small variety of distinct codes to manipulate the data as required for the mathematical functionality to properly take place. Below is an index of all the opcodes tied to instruction sets used in this lab (The Western Design Center Inc., 2018).

|  |  |  |
| --- | --- | --- |
| Opcode | Instruction | Description |
| 18 | CLC | Clear Cary flag |
| 38 | SEC | Set Carry |
| A9 | LDA | LoaD Accumulator with memory |
| 8D | STA | Store Accumulator in memory |
| 4C | JMP | JuMP to new location |
| D0 | BNE | Branch if Not Equal (Pz=0) |
| 6D | ADC | Addmemory to accumulator with Carry |
| ED | SBC | SuBtract memory from accumulator withborrow (Carry bit) |

Here we see the general approach to creating these simple math addition and subtraction functions. We have a value for a carry bit, the ability to store data in the accumulator, the ability to store accumulator data into a memory location, a way to unconditionally jump to a new memory location, a way to jump to a new memory based on the zero flag, a way to add from memory to the accumulator and include a carry bit, and lastly a way to subtract from memory while including the carry bit. All of these instruction sets are used to complete the lab.

Below is an image of the Commodore 64 Emulator’s best attempt at decoding the program that is stored in its memory between inclusive locations c000 to c070. As seen in the image there are three main sections of code split by default initialized memory locations. The first sets up the storage for the mathematical operands. The next two sections are the addition section and subtraction section. The flow of logic works through the first section. Then makes the unconditional jump to the addition section. The addition section makes a conditional check on the zero-flag set in memory and if the accumulator was last stored with data 00 it continues through the addition functionality. If, instead, the accumulator has the value FF, it makes the jump to the subtraction section of code and continues form there.  
A picture containing text, computer, screenshot, screen

Description automatically generatedA picture containing text

Description automatically generated

Visible below is the program executing the subtraction section. I discovered a mild halt in my progress while programing this lab. At first, I was storing the value of 02 in the output memory location instead of 03. After a bit of research, I realized that I did not reset the carry flag, so the subtraction function had no value to initially borrow from resulting in an incorrect output. All in all, the subtraction section does not differ much from the addition other than the carry flag problem.

A picture containing text, computer, screenshot, screen

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

Through my research I have learned that many people found assembly languages somewhat tedious to work with. Programing in basic was sometimes not quite as efficient as preferred during its early lifespan. Thankfully, people of the time period had the freedom to work with these low-level languages and make their own hybrid variants to benefit their needs (Butler, 1978). Often these relied on a base of 6510 assembly (Butler, 1978). As an example, in the very early 90s a software for training infantry on rifle handling and firing in an arcade style shooter was developed. It was primarily based on the 6510-assembly language, but still used Basic as its higher-level language support (Evans, Marshall, Wolff, Broom, & Greene, 1990). This understanding of assembly allows computer science students to better understand the operation and methodology of processors which increases proficiency in their field of study.

**References**

Butler, A. M. (1978). An Approach to Language Implementation and Code Generation for Microcomputers. *Arcade Combat Simulator (JMACS)*, 1-13.

Evans, K. L., Marshall, A. H., Wolff, R. S., Broom, J. M., & Greene, W. H. (1990). Design of a Joint Service Multipurpose Arcade Combat Simulator (JMACS). *U.S. Army Research Institute for the Behavioral and Social Sciences.*, 3-10.

The Western Design Center Inc. (2018). 65xx W65C02S 8–bit Microprocessor. *The Western Design Center Inc.*, 21-22.